SISTEMI OPERATIVI

ESERCIZIO N. 1 del 17 SETTEMBRE 2003

Un **campo da golf** ha a disposizione **P palline**. Il campo è frequentato da **giocatori esperti** e **principianti**. Gli esperti noleggiano **2** palline e hanno la priorità sui principianti; i principianti noleggiano un numero maggiore di palline, compreso tra **3** e **N** (**N** < **P**). I giocatori, una volta terminato di giocare, devono restituire il numero esatto di palline noleggiate all'inizio del gioco.
Si implementi una soluzione usando il costrutto monitor per modellare il **campo da golf** e i processi per modellare i **giocatori** e si descriva la sincronizzazione tra i processi. Nel rispettare i vincoli richiesti, si cerchi di massimizzare l'utilizzo delle risorse. Si discuta se la soluzione proposta può presentare starvation e in caso positivo per quali processi, e si propongano modifiche e/o aggiunte per evitare starvation.

```
program CampoGolf

const     P = ...; { palline }
const     N = ...; { numero max di palline per principiante }
type      gioc = (ES, PR); { tipo di giocatori }

type giocatore = process (t: gioc, q: 2..N)
begin
      repeat
            c.richiedi (t, q);
            <gioca a golf>
            c.rilascia (t, q);
      until false
end

type campo = monitor

{ variabili del monitor }
var  palledisp: integer;
      { palline disponibili }
      coda : array[gioc] of condition;
      { coda su cui sospendere i giocatori }
      gioccoda : array[gioc] of integer;
      { contatori dei giocatori in coda }

procedure entry richiedi (t: gioc, q: 2..N)
begin
      { se le palline disponibili sono meno di q }
      while palledisp < q do
      begin
            { sospensione }
            gioccoda[t] := gioccoda[t] + 1;
            coda[t].wait;
            gioccoda[t] := gioccoda[t] - 1;
      end
```

```
        { acquisizione delle risorse }
        palledisp := palledisp – q;
end

procedure entry rilascia (t: gioc, q: 2..N)
var s, i: integer;
begin
        { rilascio delle risorse }
        palledisp := palledisp + q;
        { risveglio degli esperti }
        s := gioccoda[ES];
        for i := 1 to s do
                coda[ES].signal;
        { risveglio dei principianti se non ci sono esperti in coda}
        if not coda[ES].queue then
        begin
                s := gioccoda[PR];
                for i := 1 to s do
                        coda[PR].signal;
        end
end

begin { inizializzazione delle variabili }
        palledisp := P;
        gioccoda[ES] := 0;
        gioccoda[PR] := 0;
end

var  c: campo; { il nostro monitor }
        ge1, ge2, ... : giocatore (ES, 2);
        gp1, gp2, ... : giocatore (PR, j);

begin end.
```

Starvation

Nella soluzione proposta può esserci starvation se i giocatori esperti passano sempre davanti a quelli principianti.
Si può risolvere imponendo un contatore per ogni tipo di giocatore, alternando la priorità ogni tot accessi dello stesso tipo.

Nota

Questa soluzione usa un while in sospensione e un for per risvegliare tutti i processi. Sono possibili soluzioni più raffinate in cui viene risvegliato un numero inferiore di processi. Ad esempio, sapendo il numero di palline disponibili dopo il rilascio, è possibile calcolare il numero di esperti da risvegliare.